Creating a NIEM IEPD with NIEM-UML Transcript

NIEM Program Management Office Resource

Welcome to the NIEM-UML tutorial on creating an information exchange. You should have a little background in information sharing. An understanding of NIEM, XML, and UML would be helpful, but is not really required. If you'd like more background, you may want to see the NIEM-UML high level introduction. In this tutorial we'll cover the essential elements of a NIEM-UML model, the steps for creating an IEPD from that model, and what the NIEM-UML tools produce. If you need more background, please see the NIEM-UML webinars on the NIEM.gov website.

The business case for this example will be healthcare. It is based on the design done by tony Malia using the RIM health care model as presented to the OMG's health interoperability workshop in June 2012. The subject is the exchange of healthcare medical conditions involving a client, which is a patient, and a healthcare service provider. The basic information to be shared is info about the client, which is the healthcare consumer, the service provider, the person actually providing the healthcare, the service provided to the client, by the provider, and the healthcare organization. Of course we also need to understand the names and identifiers for the people involved, the medical condition, and code lists for medical problems. For code lists we're going to use a problem code that includes finding, symptom, problem, complain, and condition. All of the above is then packaged for a healthcare information exchange.

We'll now build a model in our UML tool. First we're going to create a new project from scratch. We'll call it medical exchange. Note there are also some NIEM specific templates available. We're going to load a template that provides us with NIEM core as well as NIEM profiles. Let's review the contents of the template. Notice that we have "My NIEM" model. This is the template for us to fill in to create our IEPD. Note that inside of it we have packages for exchanges, extensions, and subsets. Packages are a way to organize UML elements in your model. Each of these packages has a corresponding diagram that we see over here on the right that has the NIEM core subset diagram. Note that this point, it's empty, as is our extension model and our exchange model. These are places to put our purpose specific content. Notice also, that the NIEM-UML profile is loaded, as

is the NIEM core data model. For those of you who are familiar with NIEM, you may recognize the rather extensive set of properties and classes associated with NIEM core. There are also diagrams for NIEM core. These allow us to explore the NIEM core model graphically.

Now let's create our specific IEPD. What we're going to do first is create an extension model to represent the high level domain concepts for information exchange. We're going to create a class for client, and one for service provider. Note the yellow halos around the classes. These are warnings. If we click on the little warning sign, we see that a NIEM naming and design rule has not been satisfied by this class. It says, the types must be documented. One thing the tool can't do for you is create your documentation unfortunately. So what we're going to do is go over here and type in some initial documentation. As we do so, note that the warnings go away meaning the NDR have been satisfied and a conformant IEPD has been produced. We also want an association between client and service provider. This is the service provided. By using an association class like this, we're able to assign properties and associations to the association itself. It's to be managed as a first class element in our model. Note also, that the ends have been named for us in accordance with typical NIEM naming conventions.

Now, what we'd like to do is connect these high level concepts with NIEM core, in particular concepts of person and organization. We're going to go into the NIEM core subset and ask the tool to show us the classifiers available. We know we want something about person, because client and service providers are people. We see there's quite a bit about person. We're going to start with person type. Then we're going to do the same thing for organization. And again, we see organization type. Note that these are brought in without properties or associations. We're going to subset NIEM core to bring in just those properties that we like (need). We'll start with organization. It lists properties that are available. We see there's quite a few for organization. At this point, we're going to bring in organization name. This is sufficient for our example; of course, you may want more for your particular application. You see it brought in organization name and that has a text type as a type. If we look down in our subset model, we see organization and person, which we selected, and also that text type has been brought in because it's required to support organization name. As we bring in properties, it brings in anything required to support those properties now let's do the same thing for person. We see a rather extensive list of properties for person. We'll just pick a few. Let's say birth date. We see that there's already some medical condition information available. We're going to bring in medical condition to see if that fits our requirement. And of course we'd want the person's name. As with organization, the properties were included, as were any of the types required, in this case, person name, and their medical condition type. What we need to do is see if there are any properties of these that we have to include as well. So we're going to look at person name first. We see that there are quite a few options representing a name. We're just going to use their full name for now. We'd also like to check out their medical condition. We're just going to bring in the condition description text. As you can see, we're starting to build up a model that represents the parts and pieces of NIEM core that we want for medical condition exchange.

But now we have to connect what we've brought in from NIEM core with our extension model. Both clients and service providers are people. We want to be able to reuse all the person type information for both clients and service provider. A good way to do that in NIEM is to use the role of association. Here we see that client is a RoleOf a person. If we want to say that a client must be the RoleOf exactly one person but a person can be a client multiple times. Likewise, a service provider is a RoleOf a person and again a person may be a service provider in multiple circumstances as represented by the cardinality from the person type to the sp. A requirement is also provided for a problem code. We're going to have a problem code enumeration and put in values for finding, symptom, complaint, and condition. We're going to then add a property to our client for this problem code. And this way we can see how we can add additional types to our model, in this case problem code, in our extension model and use them in classes that are building on NIEM core. Because this is just an example, I'm going to turn off our active validation and accept that we don't have documentation for each of these elements. One other thing we need is our actual exchange document, a condition exchange, and populate it with properties for each of the classes we've created. And there's one more thing we need to do, we need to create our exchange model to specify which of these classes can actually be a document transferred between stakeholders. For this, I'm going to create what is called a property holder because all we want in this case are properties representing the documents to be exchanged and the only document were exchanging at this point is our condition exchange.

Between the exchange model, our extension model, and our subset model, we've defined the information that will be part of our exchange. One other aspect is the metadata corresponding with the actual IEPD. In this case were going to create an IEPD for condition exchange, and know that there's already values set, these are defaults that came from our template. Our condition IEPD imports our exchange model, which is simply this one, which in turn uses our extension model which is our logical model, which in turn uses our NIEM core subset. With this information defined between what we've entered and what was provided by the template, we have everything we need for a complete IEPD. We're going to ask the tool to export this to a NIEM model package description, which in this case will be an IEPD based on the metadata provided. Let's look at what it's created. We see that the created MPD has a set of schemas, it has template for our samples, it has the NIEM libraries, it has document that's created, the change log, and catalog. All the elements required for a NIEM IEPD.

First let's look at the documentation produced based completely on what's in the model- the purpose, scope, artifacts, and detailed documentation for the IEPD is generated. Of course you'll want to add additional text but this provides a good start on the documentation for an IEPD. It includes all the required elements, the diagrams that we created in UML, and the documentation of each element whether it's in our extension model or its part of NIEM core. We're then going to look at the specific schemas produced. Non-technical users may want to skip this part. We see it created a subset model, the NIEM base models, our extension and our exchanges. Let's look at our extension model. Here we see a complete XML schema with all the elements required by the XML schema specifications as well as the NIEM Naming and Design Rules. For example, where we can

see our problem code with its specific values, we can see our condition exchange with properties for person and organization, client and sp., we see that the goal elements is required by the NIEM Naming and Design Rules has been correctly created. We see our client type and service provider type. We also see that the association between them, the service provided, has been mapped to a NIEM association type. With properties for the service provider and the client, implementing the reference scheme as required by the NIEM Naming and Design Rules. So as you can see with a relatively simple model, we've created a conformant NIEM schema and IEPD. An important part of any lifecycle is evolution.

Let's evolve our IEPD by connecting service provider with organization. Well start by creating a healthcare organization class. We now want to connect healthcare organization with organization type from NIEM core. Well say that health care organization is a role played by an organization. We can now create an association from our service provider to healthcare org, set the properties of the ends, and name them appropriately. One more thing we should do is add healthcare org to our condition exchange. This completes the update of the IEPD at the UML level. We can now export this to a NIEM MPD. This will create an updated set of schemas, documentation, and other NIEM artifacts. As a generalization is complete, we can now inspect the XML. As before, we see that the XML schema has been generated. We have a new healthcare organization type with the elements for service provider. The role, and the other details required by the NIEM Naming and Design Rules. We also see that service provider type has required property, referencing providing organization. This completes the update of the IEPD and the NIEM artifacts, as well as the tutorial on creating an IEPD with NIEM-UML.